ER622185926

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

TYPE EVOLUTION

INVENTORS:

DOUGLAS PURDY

NATASHA JETHANANDANI

SOWMY SRINIVASAN

STEFAN PHARIES

ATTORNEY'S DOCKET NO. MS1-1826US

FIELD

[0001] The present invention is directed to the interoperability of evolving types.

BACKGROUND

[0002] Type evolution, also referred to as the versioning of types, is an important aspect of a distributed environment or service oriented architecture (SOA).

[0003] Typically, an exchange of messages between two end points, such as between a server device and a client device, requires that the end points agree upon a format for the messages. A message format may otherwise be referred to as a "data structure" or a "type," and these terms may be used interchangeably throughout this description.

[0004] A type is typically described in a neutral language, a non-limiting example of which is XML Schema. Extensible Markup Language (XML) is a meta-markup language that provides a format for describing structured data. XML is a tag-based language and, by virtue of its tag-based nature, defines a strict tree structure or hierarchy. XML is a general-purpose language for representing structured data without including information that describes how to format the data for display.

[0005] XML accommodates an immeasurable number of schemas. A schema is a set of rules for constraining the structure and articulating the information set of XML messages. A "message" is formatted text, and "schema" describes the data structures, shape, and content of XML messages that are valid for a given application. As schemas evolve, data exchanges between different end points, in the form of requests and

responses are compromised. For instance, a server using a current generation of a schema may receive a request to validate a message, information, or an action from a client using a previous generation or version of the same schema, or vice-versa. The schemas used by the server and the client must be compatible with each other in order for the message, information, or action to be validated.

[0006] More particularly, for type programmers, it is expected that their types would evolve over time to add new features, delete old features, or fix bugs in existing features. Unfortunately, the interoperability of cross-generational types is uncertain without specifically programming a type to be backward-compatible or forward compatible. A backward-compatible type is able to validate messages using received data that is formatted in accordance with a previous version of the type. A forward compatible type is able to validate messages using received data that is formatted in accordance with a subsequent version of the type.

SUMMARY

[0007] Described herein are versionable types, as well as methods and means for rendering multiple generations of a type compatible with one another.

[0008] A versionable type includes an optional data member that tolerates the absence of optional data in order to validate a message. That is, a message may be validated by the versionable type despite an absent element in the optional data member. The versionable type further includes a construct that is delimited from the optional data member, which is able to accept data that is associated with another version of the type

rather than a current version of the type. Thus, a message may be validated by a versionable type despite a wildcard data element in the message.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The detailed description is described with reference to the accompanying figures.

[0010] FIG. 1 shows devices communicating over a network, with the devices implementing technologies for versioning types according to an example embodiment.

[0011] FIG. 2 shows an example embodiment of a processing flow for versioning types.

[0012] FIG. 3A shows a type according to an example embodiment.

[0013] FIG. 3B shows an example of specific elements, further to the example of FIG. 3A.

[0014] FIG. 3C shows an example of specific elements, further to the examples of FIGS. 3A and 3B.

[0015] FIG. 4 illustrates a general computer network environment which can be used to implement the techniques described herein.

DETAILED DESCRIPTION

[0016] The following description is directed to techniques for rendering different generations of types compatible with each other. More particularly, for devices such as a server device and a client device that agree upon a type before exchanging messages, the

example embodiments enable the type to evolve at one or more of the end points without adversely affecting the exchange of messages at either of the end points.

[0017] With regard to the agreement for the messages referenced above, the term "message" is used within this description to refer to a formatted message. In addition, the description may interchangeably utilize the terms "type," "schema," and "format." These terms may refer to an arrangement of data within a message file that enables a message to be encoded or decoded in a valid manner. More particularly, each of a "type," "schema," and "format" may refer to specifically arranged data members to validate a message file to be transmitted to a receiving node, or to validate a message received from a sending node. Thus, all three terms pertain to the structuring of data that is exchanged between devices including, but not limited to, a server device and a client device.

[0018] Further still, with regard to types, schemas, and formats, the description may reference "first" or "second" versions or different generations thereof. The qualifiers "first" and "second" are intended merely to distinguish different versions of a particular type, schema, or format; and are not intended to imply any sequence or order of the versions or generations thereof. For example, a second type may be a subsequent version of a first type, or it may be a preceding version of the first type. Similarly, the second type may be the immediately subsequent version of the first type, or it may succeed the first type by more than one generation; the second type may also be the immediately preceding version of the first type, or it may precede the first type by more than one generation.

[0019] FIG. 1 shows server device 105 and client device 110 that are capable of type versioning 112 in accordance with the example embodiments described herein. Server device 105, client device 110, and other data source 120, which may also be

capable of type versioning, are communicatively coupled through network 115.

[0020] Server device 105 may provide any of a variety of data and/or functionality to client device 110. The data may be publicly available or alternatively restricted, *e.g.*, restricted to only certain users or available only if an appropriate fee is paid. Server device 105 is at least one of a network server, an application server, a web blade, or any combination thereof. Other data source 120 may also be embodied by any of the above examples of server device 105. An example embodiment of server device 105 is described in further detail below with reference to FIG. 4.

[0021] Client device 110 may include any of a variety of conventional computing devices, including a desktop personal computer (PC), workstation, mainframe computer, Internet appliance, and gaming console. Further, client device 110 may be any device capable of being associated with network 115 by a wired and/or wireless link, including a personal digital assistant (PDA), laptop computer, cellular telephone, *etc.* Further still, client device 110 may include the client devices described above in various quantities and/or combinations thereof. Other data source 120 may also be embodied by any of the above examples of client device 110. An example embodiment of client device 110 is also described in further detail below with reference to FIG. 4.

[0022] Network 115 is intended to represent any of a variety of conventional network topologies, which may include any wired and/or wireless network. Network 115 may further utilize any of a variety of conventional network protocols, including public and/or proprietary protocols. For example, network 115 may include the Internet, an intranet, or at least portions of one or more local area networks (LANs).

[0023] Typically, server device 105 includes any device that is the source of content, and client device 110 includes any device that receives such content either via

network 115 or in an off-line manner. However, according to the example embodiments described herein, server device 105 and client device 110 may interchangeably be a sending host or receiving host. For the purpose of describing the example embodiments in FIGS. 2-4, the description of which is not intended to be limiting in any manner, server device 105 is a sending host and client device 110 is a receiving host. That is, for explanatory purposes only, sending host 105 transmits data that is formatted in accordance with a first type and receiving host 110 validates a message by encoding the received data in accordance with a second, versioned type.

[0024] FIG. 2 shows an example embodiment of a processing flow for versioning types. More particularly, the processing of FIG. 2 renders data functional from one type to another, or, in other words, from the first type to the second type. Versioning of types may be utilized for exchanging text, electronic messages, web-posts, *etc.*, between server device and client device 110, which utilize different types for validating messages.

[0025] According to FIG. 2, message file 205 is transmitted from sending host 105 to receiving host 110. Message file 205 is an encoded data file by which a message is to be validated at receiving host 110. Typically, such transmission of message file 205 is conducted over network 115, although the present example may also accommodate off-line transmission of message file 205.

[0026] In the example, message file 205 is encoded according to a first type at sending host 105. The example is directed towards overcoming compatibility issues that may arise when receiving host 110 receives message file 205, and attempts to use it for validating a message in accordance with a second type, which is a different version, *i.e.*, different generation, of the first type.

[0027] In particular, as will be explained below in more detail with regard to FIGS. 3A and 3B, to validate versioned message 215, message file 205 is formatted in accordance with the second type at receiving host 110. The versioning 210 of data is enabled regardless of whether message file 205 is encoded in accordance with a first type that is an earlier, later, or concurrent version of the second type at receiving host 110. Further, the second type is able to validate versioned message 215 regardless of whether the first type precedes or succeeds the second type by one or more generation. In other words, the versioning 210 of data is enabled because the second type at receiving host 110 is simultaneously backward compatible and forward compatible.

[0028] FIG. 3A shows an example embodiment of the aforementioned second type that is simultaneously backward-compatible and forward-compatible. With regard to FIGS. 3A and 3B, the second type format may alternatively be referred to as a "current type version" or a variation thereof, since the referenced type is being utilized at receiving host 110 to validate a message based on the data contained in message file 205. More particularly, data from message file 205 is inserted into appropriate members or constructs of second type 305 to validate a message even if sending host 105 and receiving host 110 utilize different versions, or generations, of type 305.

[0029] Type 305 includes data members F1 310 and F2 315 for receiving data from message file 205. Data members F1 310 and F2 315 are structural constructs for receiving data used to validate a message in accordance with type 305 at receiving host 110. The data entity corresponding to the data to be received in data members F1 310 and F2 315 may be required or optional for validating a message in accordance with type 305. With regard to the present example, optional data refers to a known data entity that may or may not be required for a message to be validated.

[0030] If the data entity corresponding to the data to be received in data members F1 310 and F2 315 is required, type 305 is unable to validate a message if data for either of data members F1 310 and F2 315 is absent from message file 205. For instance, consider an example in which message file 205 is encoded in accordance with a first type at sending host 105. When message file 205 does not include data corresponding to data member F1 310 and/or data member F2 315, a message cannot be validated by type 305 at receiving host 110.

[0031] If a data entity corresponding to data for either of data members F1 310 and F2 315 is deemed to be optional, type 305 may tolerate an absence of such data and still validate a message at receiving host 110. That is, whether the first type by which message file is encoded at sending host 105 is the same or different than type 305 at receiving host 110, a message may be validated by type 305 at receiving host 110 when the data member corresponding to an optional data entity is empty.

[0032] If type 305 is described by an XML schema, an appropriate designation (*e.g.*, "minOccurs = 0") is made for an optional data entity corresponding to either of data members F1 310 and F2 315. More particularly, the designation indicates that an absent data element corresponding to the optional data entity may be tolerated. Otherwise, without such designation, it may be assumed that the data entity corresponding to data members F1 310 and F2 315 is required to validate a message in accordance with type 305.

[0033] Type 305 further includes constructs 320A and 320B for receiving data from message file 205. According to the example embodiments described herein, constructs contain a delimiter followed by an open element. More specifically, constructs 320A and 320B are structural constructs for receiving data used to validate a message

based on message file 205, which is encoded in accordance with a first type that is a different version of type 305 utilized at receiving host 110. The different type may be either a preceding or subsequent version of type 305. To be both backward-compatible and forward-compatible, type 305 may include at least one construct 320A, which is unbounded with regard to a number of occurrences within type 305. Thus, construct 320A is able to have multiple constructs nested therein, including construct 320B, with each successive construct enabling the occurrence of at least a further construct. That is, an occurrence of a construct enables the occurrence of a subsequent construct, until a terminating sentry (*i.e.*, an end delimiter) is detected.

[0034] The example of FIG. 3A includes two constructs 320A and 320B and is provided only as a non-limiting example. Rather, constructs 320A and 320B are provided to illustrate both the multi-generational, backward- and forward-compatible characteristics of type 305. Further, although the example of FIG. 3A shows constructs 320A and 320B at the end of type 305, a construct may be inserted at any place within a type that may accommodate a schema particle.

[0035] Constructs 320A and 320B may be alternately referred to as "placeholders." Construct 320A contains delimiter 325, followed by wildcard member 330; and construct 320B contains delimiter 335, followed by wildcard member 340. End delimiter 345 denotes an end to the wildcard members in message file 205. With regard to the present example, a wildcard member is a data member that receives data for an optional and/or unknown data entity that may be used to validate a message in accordance with type 305. Thus, a wildcard member is essentially an optional data member, although an optional data member is not necessarily a wildcard member.

[0036] Each of delimiters 325 and 335 is utilized as a sentry to validate the beginning of the respective construct. In addition, the delimiters render the respective constructs deterministic, particularly as they follow an optional data member, adhering to a fundamental principle for maintaining the determinism of an XML schema. Thus, when the data entity corresponding to data member F2 315 is optional, delimiter 325 renders construct 320A deterministic as construct 320A follows optional data member F2 315; and delimiter 325 further renders construct 320B deterministic as construct 320B follows wildcard member F3 330.

[0037] Each of the wildcard members 330 and 340 for the respective constructs may accept data from message file 205 that is optional data (*i.e.*, from a known data entity) or wildcard data (*i.e.*, from a data entity that is unknown by type 305). Further, each construct is to be appropriately annotated as being unbounded in the number of further constructs nested therein, dependent upon the amount of data contained in message file 205. End delimiter 345 denotes an end to wildcard members in message file 205.

[0038] Further to the example of type 305 in FIG. 3A, construct 320A includes delimiter 325 and wildcard member F3 330. The example contemplates message file 205 being encoded at sending host 105 in accordance with a first type, with the first type being a different version of the type 305 at receiving host 110. The different version of type 305 may be a version preceding or succeeding type 305 by one or more generations.

[0039] As set forth above, a wildcard member is an optional data member. Thus, wildcard member F3 330 is able to tolerate the absence of a data corresponding to a known data entity from message file 205, if such data entity has been deemed to be

optional by type 305. Alternatively, wildcard member 330 is able to receive such optional data that is contained in message file 205.

[0040] Wildcard member F3 330 is also able to receive wildcard data from message file 205. That is, when message file 205 is encoded according to the first type at sending host 105, and the first type includes data corresponding to a data entity that is unknown to type 305 at receiving host 110, wildcard member F3 330 is able to receive data corresponding to the unknown data entity in order for type 305 to validate a message at receiving host 110.

[0041] Construct 320B, which is nested in construct 320A, is provided to receive further wildcard data from message file 205. Construct 320B includes delimiter 335 to maintain the deterministic character of type 305 since the preceding wildcard member F3 330 is also an optional data member. Wildcard member 340 is further provided to receive data corresponding to an unknown data entity in order for type 305 to validate a message at receiving host 110. In addition, construct 320B may have further constructs nested therein. That is, the occurrence of still further constructs may be enabled until a terminating sentry is detected.

[0042] Accordingly, receiving host 110 is able to receive message file 205 and validate a message in accordance with type 305, even if message file 205 is encoded in accordance with a different version of type 305. More particularly, a message may be validated in accordance with type 305 regardless of whether the different version of type 305 by which message file 205 is encoded precedes or succeeds type 305 by one or more generation.

[0043] FIG. 3B shows a detailed example of constructs 320A and 320B corresponding to type 305 in FIG. 3A. More specifically, FIG. 3B shows an example of

an XML schema as type 305. FIG. 3B does not show schema elements for data members F1 310 and F2 315, since a description of such data members may be attributed to fundamental principles of XML that are known in the art. This example is not intended to be limiting in any manner, nor should any limitations be inferred from FIG. 3B.

[0044] It is first noted that XML is a tag-based language, and thus XML defines a strict tree structure or hierarchy. XML enables structured data to be described and exchanged in an open, text-based format. Further, XML utilizes the concepts of elements and namespaces. XML is a general-purpose language for representing structured data without including information that describes how to format the data for display. XML "elements" are structural constructs that consist of a start tag, an end or close tag, and the information or content that is contained between the tags. A "start tag" is formatted as "<tagname>" and an "end tag" is formatted as "</tagname>". In an XML message, start and end tags can be nested within other start and end tags. All elements that occur within a particular element must have their start and end tags occur before the end tag of that particular element. This defines a tree-like structure that is representative of the XML message. Each element forms a node in this tree, and potentially has "child" or "branch" nodes. The child nodes represent any XML elements that occur between the start and end tags of the "parent" node. Further, the unique particle attribution (UPA) principle advocates that each element in a message be attributed to only one schema tag.

[0045] In FIG. 3B, construct 320A is introduced by line 350 <xs:sequence minOccurs = "0" maxOccurs = "1"> indicating that the sequence that follows is optional, and may occur only once.

[0046] Line 355 <xs:element ref="beginLocal" minOccurs = "1" maxOccurs = "1"/> corresponds to delimiter 325 of construct 320A in FIG. 3A. The delimiter is called

“beginLocal” and “ref” indicates that the delimiter is from a different namespace than the data that follows. Further, the delimiter may occur a minimum of one time and a maximum of one time.

[0047] Line 360 <xs:element name="v2LocalField" type="xs:int" /> indicates an element to be named “v2LocalField.”

[0048] Line 365 <xs:sequence minOccurs="0" maxOccurs="unbounded"/> denotes optional member F3 of construct 320A, and indicates that the sequence that follows is optional (by “minOccurs = 0) and may occur any number of times (by “maxOccurs = unbounded).

[0049] Line 370 <xs:element ref ="beginLocal" minOccurs = "1" maxOccurs = "1" /> denotes delimiter 335 in construct 320B.

[0050] Line 375 <xs:any namespace = "##targetNamespace ##local" minOccurs="0" maxOccurs= "unbounded"/> corresponds to wildcard member 340 of construct 320B. The wildcard elements corresponding to “targetNamespace” are optional (by “minOccurs=0”) and unbounded in the number of occurrences that are possible (by “maxOccurs=0”). Namespace ##targetNamespace denotes that the wildcard elements are in the namespace corresponding to the type. Namespace ##local denotes that the wildcard elements may corresponding to an empty namespace. It is further noted that construct 320B may not occur without the occurrence of construct 320A. That is, nested construct 320B may only occur after the occurrence of the preceding construct 320A.

[0051] Line 380 </xs:sequence> ends the sequence of nested constructs.

[0052] Line 385 <xs:element ref="endLocal" minOccurs = "1" maxOccurs = "1" /> corresponds to end delimiter 345 in FIG. 3A indicating the end of local namespace entries.

[0053] Line 390 </xs:sequence> ends the schema.

[0054] FIG. 3C shows sequence 307, which provides constructs further to the example of FIG. 3B. More specifically, the sequence of FIG. 3C allows elements from a target namespace or local namespace, which corresponds to sequence 305 from FIG. 3B. To be able to insert global elements, sequence 306 is provided. The constructs of sequence 306 are enabled in the same manner as the constructs of FIG. 3B.

[0055] By the example embodiments described above, a versionable schema is presented that is simultaneously backward- and forward- compatible with different versions of a schema. The versionable schema is able to tolerate the absence of optional data, and further accept wildcard data from unexpected versions of the schema. Thus, a message or message may be validated by inserting the received data into the versionable schema.

[0056] FIG. 4 illustrates a general computer environment 400, which can be used to implement the techniques described herein. The computer environment 400 is only one example of a computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the computer and network architectures. Neither should the computer environment 400 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the example computer environment 400.

[0057] Computer environment 400 includes a general-purpose computing device in the form of a computer 402. Sending host 105 and receiving host 110 for versioning types in accordance with the example embodiments described above may be included as various embodiments of computer 402. The components of computer 402 can include, but are not limited to, one or more processors or processing units 404, system memory

406, and system bus 408 that couples various system components including processor 404 to system memory 406.

[0058] System bus 408 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. By way of example, such architectures can include an Industry Standard Architecture (ISA) bus, a Micro Channel Architecture (MCA) bus, an Enhanced ISA (EISA) bus, a Video Electronics Standards Association (VESA) local bus, a Peripheral Component Interconnects (PCI) bus also known as a Mezzanine bus, a PCI Express bus, a Universal Serial Bus (USB), a Secure Digital (SD) bus, or an IEEE 1394, i.e., FireWire, bus.

[0059] Computer 402 may include a variety of computer readable media. Such media can be any available media that is accessible by computer 402 and includes both volatile and non-volatile media, removable and non-removable media.

[0060] System memory 406 includes computer readable media in the form of volatile memory, such as random access memory (RAM) 410; and/or non-volatile memory, such as read only memory (ROM) 412 or flash RAM. Basic input/output system (BIOS) 414, containing the basic routines that help to transfer information between elements within computer 402, such as during start-up, is stored in ROM 412 or flash RAM. RAM 410 typically contains data and/or program modules that are immediately accessible to and/or presently operated on by processing unit 404.

[0061] Computer 402 may also include other removable/non-removable, volatile/non-volatile computer storage media. By way of example, FIG. 4 illustrates hard disk drive 416 for reading from and writing to a non-removable, non-volatile magnetic media (not shown), magnetic disk drive 418 for reading from and writing to removable,

non-volatile magnetic disk 420 (e.g., a "floppy disk"), and optical disk drive 422 for reading from and/or writing to a removable, non-volatile optical disk 424 such as a CD-ROM, DVD-ROM, or other optical media. Hard disk drive 416, magnetic disk drive 418, and optical disk drive 422 are each connected to system bus 408 by one or more data media interfaces 425. Alternatively, hard disk drive 416, magnetic disk drive 418, and optical disk drive 422 can be connected to the system bus 408 by one or more interfaces (not shown).

[0062] The disk drives and their associated computer-readable media provide non-volatile storage of computer readable instructions, data structures, program modules, and other data for computer 402. Although the example illustrates a hard disk 416, removable magnetic disk 420, and removable optical disk 424, it is appreciated that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes or other magnetic storage devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or other optical storage, random access memories (RAM), read only memories (ROM), electrically erasable programmable read-only memory (EEPROM), and the like, can also be utilized to implement the example computing system and environment.

[0063] Any number of program modules can be stored on hard disk 416, magnetic disk 420, optical disk 424, ROM 412, and/or RAM 410, including by way of example, operating system 426, one or more application programs 428, other program modules 430, and program data 432. Each of such operating system 426, one or more application programs 428, other program modules 430, and program data 432 (or some combination thereof) may implement all or part of the resident components that support the distributed file system.

[0064] A user can enter commands and information into computer 402 via input devices such as keyboard 434 and a pointing device 436 (e.g., a "mouse"). Other input devices 438 (not shown specifically) may include a microphone, joystick, game pad, satellite dish, serial port, scanner, and/or the like. These and other input devices are connected to processing unit 404 via input/output interfaces 440 that are coupled to system bus 408, but may be connected by other interface and bus structures, such as a parallel port, game port, or a universal serial bus (USB).

[0065] Monitor 442 or other type of display device can also be connected to the system bus 408 via an interface, such as video adapter 444. In addition to monitor 442, other output peripheral devices can include components such as speakers (not shown) and printer 446 which can be connected to computer 402 via I/O interfaces 440.

[0066] Computer 402 can operate in a networked environment using logical connections to one or more remote computers, such as remote computing device 448. By way of example, remote computing device 448 can be a PC, portable computer, a server (such as server device 105), a router, a network computer, a peer device or other common network node, and the like. Remote computing device 448 is illustrated as a portable computer that can include many or all of the elements and features described herein relative to computer 402. Alternatively, computer 402 can operate in a non-networked environment as well.

[0067] Logical connections between computer 402 and remote computer 448 are depicted as a local area network (LAN) 450 and a general wide area network (WAN) 452. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

[0068] When implemented in a LAN networking environment, computer 402 is connected to local network 450 via network interface or adapter 454. When implemented in a WAN networking environment, computer 402 typically includes modem 456 or other means for establishing communications over wide network 452. Modem 456, which can be internal or external to computer 402, can be connected to system bus 408 via I/O interfaces 440 or other appropriate mechanisms. It is to be appreciated that the illustrated network connections are examples and that other means of establishing at least one communication link between computers 402 and 448 can be employed.

[0069] In a networked environment, such as that illustrated with computing environment 400, program modules depicted relative to computer 402, or portions thereof, may be stored in a remote memory storage device. By way of example, remote application programs 458 reside on a memory device of remote computer 448. For purposes of illustration, applications or programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of computing device 402, and are executed by at least one data processor of the computer.

[0070] Various modules and techniques may be described herein in the general context of computer-executable instructions, such as program modules, executed by one or more computers or other devices. Generally, program modules include routines, programs, objects, components, data structures, etc. for performing particular tasks or implement particular abstract data types. Typically, the functionality of the program modules may be combined or distributed as desired in various embodiments.

[0071] An implementation of these modules and techniques may be stored on or transmitted across some form of computer readable media. Computer readable media can be any available media that can be accessed by a computer. By way of example, and not limitation, computer readable media may comprise "computer storage media" and "communications media."

[0072] "Computer storage media" includes volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules, or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by a computer.

[0073] "Communication media" typically embodies computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as carrier wave or other transport mechanism. Communication media also includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. As a non-limiting example only, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared, and other wireless media. Combinations of any of the above are also included within the scope of computer readable media.

[0074] Reference has been made throughout this specification to "one embodiment," "an embodiment," or "an example embodiment" meaning that a particular

described feature, structure, or characteristic is included in at least one embodiment of the present invention. Thus, usage of such phrases may refer to more than just one embodiment. Furthermore, the described features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

[0075] One skilled in the relevant art may recognize, however, that the invention may be practiced without one or more of the specific details, or with other methods, resources, materials, etc. In other instances, well known structures, resources, or operations have not been shown or described in detail merely to avoid obscuring aspects of the invention.

[0076] While example embodiments and applications of the present invention have been illustrated and described, it is to be understood that the invention is not limited to the precise configuration and resources described above. Various modifications, changes, and variations apparent to those skilled in the art may be made in the arrangement, operation, and details of the methods and systems of the present invention disclosed herein without departing from the scope of the claimed invention.